Wine_clustering

July 25, 2023

1 Determining the group of wines using K-Means clustering

It's presented data that are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines.

The attributes are:

1) Alcohol
2) Malic acid
3) Ash
4) Alcalinity of ash
5) Magnesium
6) Total phenols
7) Flavanoids
8) Nonflavanoid phenols
9) Proanthocyanins
10) Color intensity
11) Hue
12) OD280/OD315 of diluted wines
13) Proline

(These attributes were dontated by Riccardo Leardi, riclea@anchem.unige.it)

1.1 Data source:

This dataset was provided by Stefan Aeberhard and M. Forina. It can be accessed from the UC Irvin Machine Learning Repository

1.2 Load libraries:

[95]:
```
# Import standard operational packages.
import numpy as np
import pandas as pd

# Important tools for modeling and evaluation.
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Import visualization packages.
import matplotlib.pyplot as plt
import seaborn as sns
```

1.3 Load data:

[96]:
```
col_names = ['Class_id', 'Alcohol', 'Malic_acid', 'Ash', 'Alcalinity_of_ash',␣
↪'Magnesium', 'Total_phenols', 'Flavanoids', 'Nonflavanoid_phenols',␣
↪'Proanthocyanins', 'Color_intensity', 'Hue', 'OD280/OD315_of_diluted wines',␣
↪'Proline']

wines_df = pd.read_csv("wine.data", names = col_names)
```

2 EDA:

The exploratory data analysis will help us understanding our data.

2.1 Data inspection:

[97]:
```
wines_df.head(10)
```

[97]:
```
   Class_id  Alcohol  Malic_acid   Ash  Alcalinity_of_ash  Magnesium  \
0         1    14.23        1.71  2.43               15.6        127
1         1    13.20        1.78  2.14               11.2        100
2         1    13.16        2.36  2.67               18.6        101
3         1    14.37        1.95  2.50               16.8        113
4         1    13.24        2.59  2.87               21.0        118
5         1    14.20        1.76  2.45               15.2        112
6         1    14.39        1.87  2.45               14.6         96
7         1    14.06        2.15  2.61               17.6        121
8         1    14.83        1.64  2.17               14.0         97
9         1    13.86        1.35  2.27               16.0         98

   Total_phenols  Flavanoids  Nonflavanoid_phenols  Proanthocyanins  \
0           2.80        3.06                  0.28             2.29
```

```
1          2.65        2.76                  0.26              1.28
2          2.80        3.24                  0.30              2.81
3          3.85        3.49                  0.24              2.18
4          2.80        2.69                  0.39              1.82
5          3.27        3.39                  0.34              1.97
6          2.50        2.52                  0.30              1.98
7          2.60        2.51                  0.31              1.25
8          2.80        2.98                  0.29              1.98
9          2.98        3.15                  0.22              1.85

   Color_intensity   Hue  OD280/OD315_of_diluted wines  Proline
0             5.64  1.04                          3.92     1065
1             4.38  1.05                          3.40     1050
2             5.68  1.03                          3.17     1185
3             7.80  0.86                          3.45     1480
4             4.32  1.04                          2.93      735
5             6.75  1.05                          2.85     1450
6             5.25  1.02                          3.58     1290
7             5.05  1.06                          3.58     1295
8             5.20  1.08                          2.85     1045
9             7.22  1.01                          3.55     1045
```

[98]:
```
wines_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 14 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Class_id                      178 non-null    int64
 1   Alcohol                       178 non-null    float64
 2   Malic_acid                    178 non-null    float64
 3   Ash                           178 non-null    float64
 4   Alcalinity_of_ash             178 non-null    float64
 5   Magnesium                     178 non-null    int64
 6   Total_phenols                 178 non-null    float64
 7   Flavanoids                    178 non-null    float64
 8   Nonflavanoid_phenols          178 non-null    float64
 9   Proanthocyanins               178 non-null    float64
 10  Color_intensity               178 non-null    float64
 11  Hue                           178 non-null    float64
 12  OD280/OD315_of_diluted wines  178 non-null    float64
 13  Proline                       178 non-null    int64
dtypes: float64(11), int64(3)
memory usage: 19.6 KB
```

[99]:
```
wines_df.describe()
```

[99]:

	Class_id	Alcohol	Malic_acid	Ash	Alcalinity_of_ash
count	178.000000	178.000000	178.000000	178.000000	178.000000
mean	1.938202	13.000618	2.336348	2.366517	19.494944
std	0.775035	0.811827	1.117146	0.274344	3.339564
min	1.000000	11.030000	0.740000	1.360000	10.600000
25%	1.000000	12.362500	1.602500	2.210000	17.200000
50%	2.000000	13.050000	1.865000	2.360000	19.500000
75%	3.000000	13.677500	3.082500	2.557500	21.500000
max	3.000000	14.830000	5.800000	3.230000	30.000000

	Magnesium	Total_phenols	Flavanoids	Nonflavanoid_phenols
count	178.000000	178.000000	178.000000	178.000000
mean	99.741573	2.295112	2.029270	0.361854
std	14.282484	0.625851	0.998859	0.124453
min	70.000000	0.980000	0.340000	0.130000
25%	88.000000	1.742500	1.205000	0.270000
50%	98.000000	2.355000	2.135000	0.340000
75%	107.000000	2.800000	2.875000	0.437500
max	162.000000	3.880000	5.080000	0.660000

	Proanthocyanins	Color_intensity	Hue
count	178.000000	178.000000	178.000000
mean	1.590899	5.058090	0.957449
std	0.572359	2.318286	0.228572
min	0.410000	1.280000	0.480000
25%	1.250000	3.220000	0.782500
50%	1.555000	4.690000	0.965000
75%	1.950000	6.200000	1.120000
max	3.580000	13.000000	1.710000

	OD280/OD315_of_diluted wines	Proline
count	178.000000	178.000000
mean	2.611685	746.893258
std	0.709990	314.907474
min	1.270000	278.000000
25%	1.937500	500.500000
50%	2.780000	673.500000
75%	3.170000	985.000000
max	4.000000	1680.000000

[100]:
```
wines_df.shape
```

[100]: (178, 14)

[101]:
```
wines_df.size
```

[101]: 2492

2.2 Data cleaning:

An assumption of K-means is that there are no missing values. Lets check for missing values in the rows of the data.

[102]:
```
wines_df.isnull().sum()
```

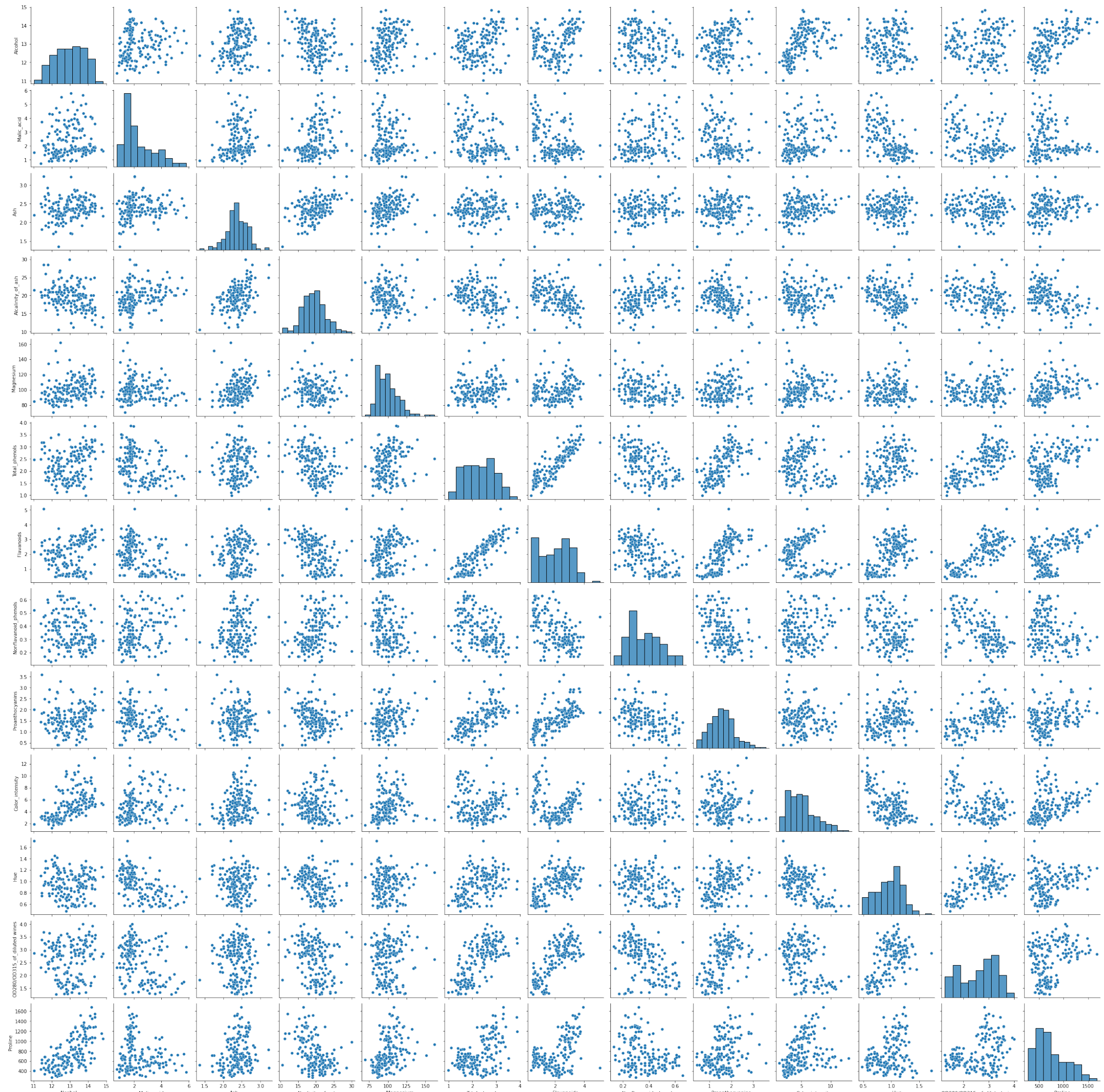
[102]:
```
Class_id                        0
Alcohol                         0
Malic_acid                      0
Ash                             0
Alcalinity_of_ash               0
Magnesium                       0
Total_phenols                   0
Flavanoids                      0
Nonflavanoid_phenols            0
Proanthocyanins                 0
Color_intensity                 0
Hue                             0
OD280/OD315_of_diluted wines    0
Proline                         0
dtype: int64
```

2.3 Data visualization:

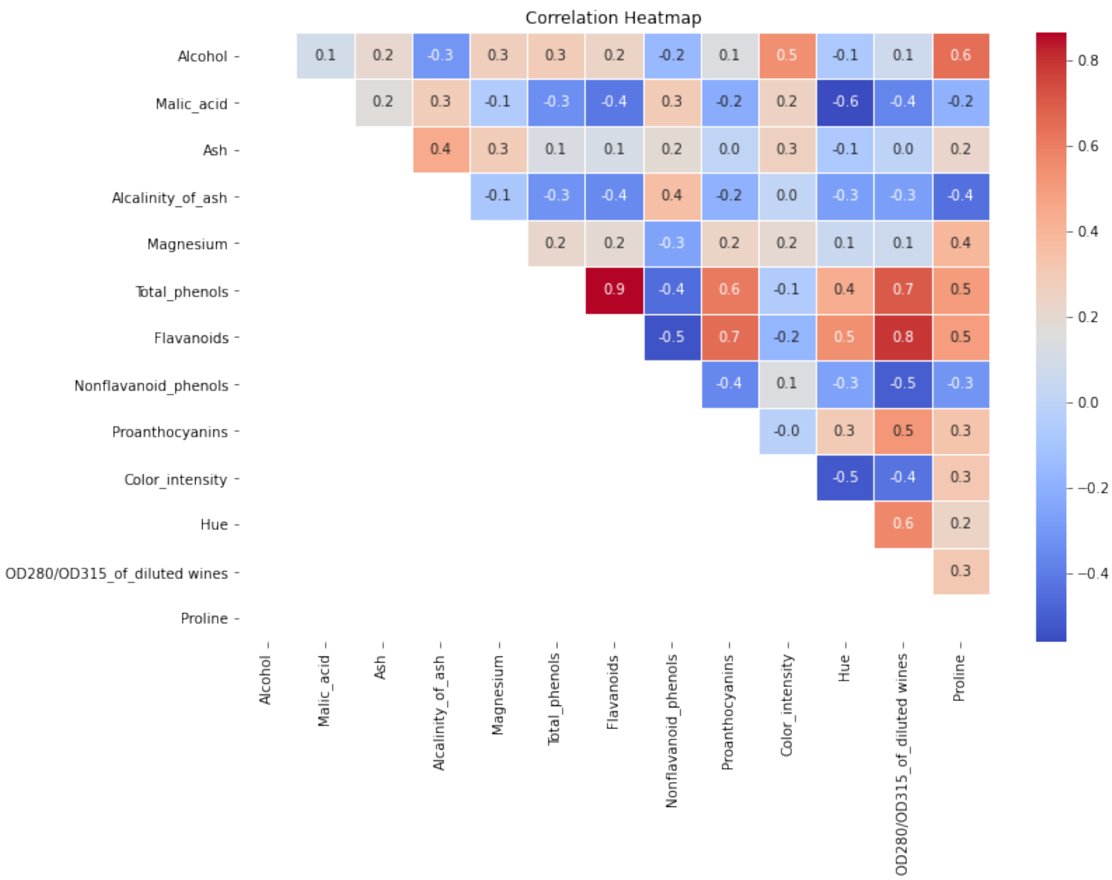
Lets check if there is any correlation between variables. We exclude the first column, it's not relevant.

[103]:
```
sns.pairplot(wines_df.iloc[:,1:], hue=None, palette='Viridis')
```

[103]:
```
<seaborn.axisgrid.PairGrid at 0x25ab60b8c10>
```



[104]:
```
corr_matrix = wines_df.iloc[:,1:].corr()
mask= np.tril(corr_matrix)

plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".1f", linewidths=0.
 ↪5, mask=mask)
plt.title("Correlation Heatmap")
plt.show()
```



2.4 Feature engineering:

Lets preparare our data before clustering.

2.4.1 Feature selection:

[105]:
```
wines_subset = wines_df.drop(['Class_id'], axis = 1)

wines_subset.head()
```

[105]:
```
   Alcohol  Malic_acid   Ash  Alcalinity_of_ash  Magnesium  Total_phenols  \
0    14.23        1.71  2.43               15.6        127           2.80
1    13.20        1.78  2.14               11.2        100           2.65
2    13.16        2.36  2.67               18.6        101           2.80
3    14.37        1.95  2.50               16.8        113           3.85
4    13.24        2.59  2.87               21.0        118           2.80

   Flavanoids  Nonflavanoid_phenols  Proanthocyanins  Color_intensity   Hue  \
0        3.06                  0.28             2.29             5.64  1.04
1        2.76                  0.26             1.28             4.38  1.05
```

```
2         3.24                    0.30              2.81              5.68   1.03
3         3.49                    0.24              2.18              7.80   0.86
4         2.69                    0.39              1.82              4.32   1.04

   OD280/OD315_of_diluted wines  Proline
0                          3.92     1065
1                          3.40     1050
2                          3.17     1185
3                          3.45     1480
4                          2.93      735
```

2.4.2 Feature transformation:

Because K-means uses distance between observations as its measure of similarity, it's important to scale the data before modeling.

[106]:
```
scaler = StandardScaler().fit(wines_subset)

wines_subset_scaled = scaler.transform(wines_subset)
```

3 Data modeling:

3.1 Evaluate Inertia

Because we don't know how many clusters exist in the data, lets start by fitting K-means and examining the inertia values for different values of k.

[107]:
```
def kmeans_inertia(k, X):
        '''
    Fits a KMeans model for different values of k.
    Calculates an inertia score for each k value.

    Args:
        k: (list of ints) - The different k values to try
        X: (array) - The training data

    Returns:
        inertia: (list) - A list of inertia scores, one for each value of k
    '''

        inertia = []

        for i in k:
            kms = KMeans(n_clusters = i, random_state = 42)
            kms.fit(X)
            inertia.append(kms.inertia_)

        return inertia
```

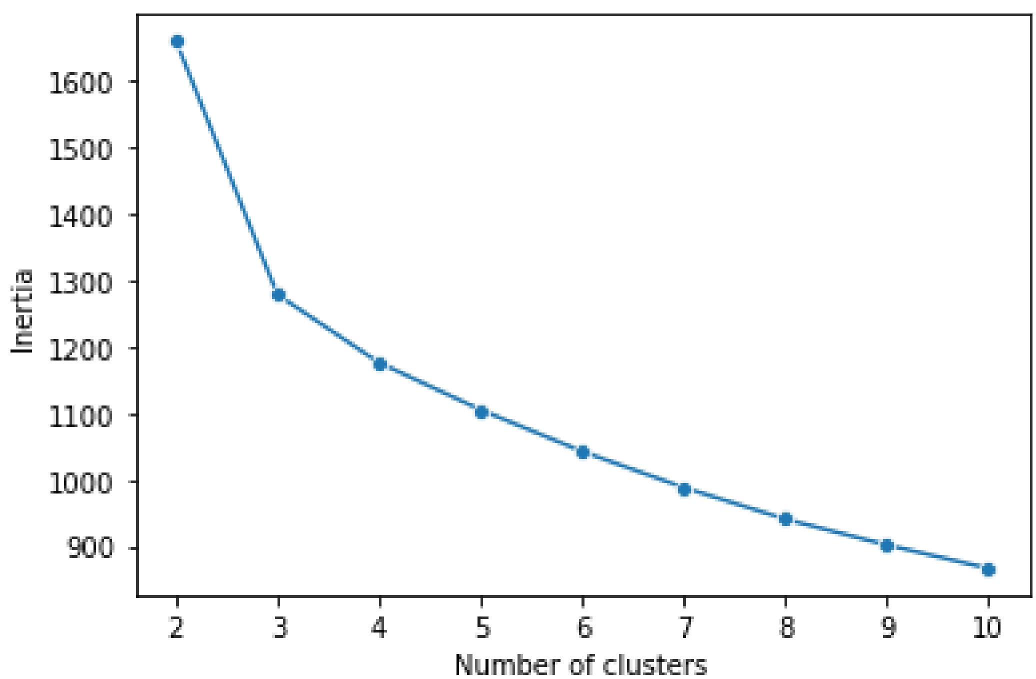
[108]:
```
num_clusters = [i for i in range(2, 11)]

Inertia = kmeans_inertia(num_clusters, wines_subset_scaled)

Inertia
```

[108]:
```
[1659.0079672511504,
 1277.928488844643,
 1175.7051928197127,
 1104.861683962532,
 1042.3872037251417,
 988.0533283180057,
 940.708165089653,
 902.0783170433883,
 866.7991687164842]
```

3.2 Elbow method

We use the elbow method to find the optimal number of clusters. Plotting the inertia values in a simple line graph with the k values along the x-axis, we could see an "elbow", which is usually the part of the curve with the sharpest angle.

[109]:
```
plot = sns.lineplot(x=num_clusters, y=Inertia, marker = 'o')
plot.set_xlabel("Number of clusters");
plot.set_ylabel("Inertia");
```



The plot seems to depict an elbow at 3 clusters, but there isn't a clear method for confirming that a three-cluster model is optimal. Therefore, we'll check the silhouette scores.

3.3 Evaluate Silhouette scores

Silhouette score provide insights as to what the optimal value for k should be, and uses both intracluster and intercluster measurements in its calculations.

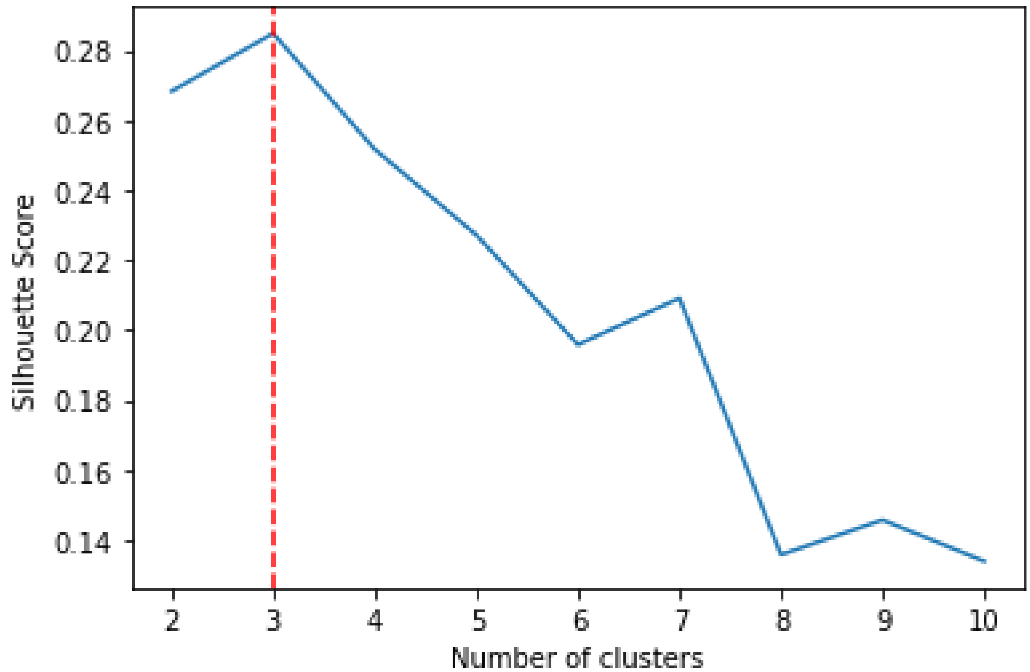
[142]:
```
def kmeans_sil(k, X):
    '''
    Fits a KMeans model for different values of k.
    Calculates a silhouette score for each k value

    Args:
        k: (list of ints) - The different k values to try
        X: (array) - The training data

    Returns:
        sil_scores: (list) - A list of silhouette scores, one for each value of␣
 ↪k
    '''

    sil_scores = []

    for i in k:
        kms = KMeans(n_clusters = i, random_state = 42)
        kms.fit(X)
        sil_scores.append(silhouette_score(X, kms.labels_))

    return sil_scores
```

[111]:
```
sil_scores = kmeans_sil(num_clusters,wines_subset_scaled)

sil_scores
```

[111]:
```
[0.26831340971052126,
 0.2848589191898987,
 0.25173343011696475,
 0.2271732547624458,
 0.19582485390848947,
 0.20913005310687274,
 0.13581656516941268,
 0.14576057110571292,
 0.13394527355239233]
```

We can plot the silhouette score for each value of k, just as we did for inertia. However, for silhouette score, greater numbers (closest to 1) are better, so we hope to see at least one clear "peak" that is close to 1.

[112]:
```
plot = sns.lineplot(x=num_clusters, y=sil_scores)
plot.axvline(x=3, color='red', linestyle='--')
plot.set_xlabel("Number of clusters");
plot.set_ylabel("Silhouette Score");
```



This plot indicates that the silhouette score is closest to 1 when our data is partitioned into **three clusters**. It confirms what we saw in the inertia analysis, where we noticed an elbow where k=3.

3.4 K-means Clustering Model

At this point, we'll instantiate a new K-means model with 3 clusters and fit it to our data.

[113]:
```
KMeans_3 = KMeans(n_clusters=3, random_state=42)
KMeans_3.fit(wines_subset_scaled)
```

[113]:
```
KMeans(n_clusters=3, random_state=42)
```

[114]:
```
print(KMeans_3.labels_[:])
print('Unique labels:', np.unique(KMeans_3.labels_))
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 0 2 2 2 2 2 2 2 2 2 2 2 1
 2 2 2 2 2 2 2 2 2 0 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 0 2 2 1 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
Unique labels: [0 1 2]
```

We can assign a new column to the original unscaled dataframe with the cluster assignment from the final K-means model.

[115]:
```
wines_df['Cluster'] = KMeans_3.labels_
wines_df.head(-10)
```

[115]:
```
     Class_id  Alcohol  Malic_acid   Ash  Alcalinity_of_ash  Magnesium  \
0           1    14.23        1.71  2.43               15.6        127
1           1    13.20        1.78  2.14               11.2        100
2           1    13.16        2.36  2.67               18.6        101
3           1    14.37        1.95  2.50               16.8        113
4           1    13.24        2.59  2.87               21.0        118
..        …        …         …   …                 …        …
163         3    12.96        3.45  2.35               18.5        106
164         3    13.78        2.76  2.30               22.0         90
165         3    13.73        4.36  2.26               22.5         88
166         3    13.45        3.70  2.60               23.0        111
167         3    12.82        3.37  2.30               19.5         88

     Total_phenols  Flavanoids  Nonflavanoid_phenols  Proanthocyanins  \
0             2.80        3.06                  0.28             2.29
1             2.65        2.76                  0.26             1.28
2             2.80        3.24                  0.30             2.81
3             3.85        3.49                  0.24             2.18
4             2.80        2.69                  0.39             1.82
..               …           …                   …                …
163           1.39        0.70                  0.40             0.94
164           1.35        0.68                  0.41             1.03
165           1.28        0.47                  0.52             1.15
166           1.70        0.92                  0.43             1.46
167           1.48        0.66                  0.40             0.97

     Color_intensity   Hue  OD280/OD315_of_diluted wines  Proline  Cluster
0               5.64  1.04                          3.92     1065        1
1               4.38  1.05                          3.40     1050        1
2               5.68  1.03                          3.17     1185        1
3               7.80  0.86                          3.45     1480        1
4               4.32  1.04                          2.93      735        1
..                 …    …                            …      …        …
163             5.28  0.68                          1.75      675        0
164             9.58  0.70                          1.68      615        0
165             6.62  0.78                          1.75      520        0
166            10.68  0.85                          1.56      695        0
167            10.26  0.72                          1.75      685        0

[168 rows x 15 columns]
```

Lets verify if it assigned each point to the right group of wine, comparing with the first column removed at the beginning.

[116]:
```
wines_df.head(-5)
```

[116]:
```
     Class_id  Alcohol  Malic_acid   Ash  Alcalinity_of_ash  Magnesium  \
0           1    14.23        1.71  2.43               15.6        127
1           1    13.20        1.78  2.14               11.2        100
2           1    13.16        2.36  2.67               18.6        101
3           1    14.37        1.95  2.50               16.8        113
4           1    13.24        2.59  2.87               21.0        118
..        …        …           …   …                …          …
168         3    13.58        2.58  2.69               24.5        105
169         3    13.40        4.60  2.86               25.0        112
170         3    12.20        3.03  2.32               19.0         96
171         3    12.77        2.39  2.28               19.5         86
172         3    14.16        2.51  2.48               20.0         91

     Total_phenols  Flavanoids  Nonflavanoid_phenols  Proanthocyanins  \
0             2.80        3.06                  0.28             2.29
1             2.65        2.76                  0.26             1.28
2             2.80        3.24                  0.30             2.81
3             3.85        3.49                  0.24             2.18
4             2.80        2.69                  0.39             1.82
..             …          …                   …               …
168           1.55        0.84                  0.39             1.54
169           1.98        0.96                  0.27             1.11
170           1.25        0.49                  0.40             0.73
171           1.39        0.51                  0.48             0.64
172           1.68        0.70                  0.44             1.24

     Color_intensity   Hue  OD280/OD315_of_diluted wines  Proline  Cluster
0           5.640000  1.04                          3.92     1065        1
1           4.380000  1.05                          3.40     1050        1
2           5.680000  1.03                          3.17     1185        1
3           7.800000  0.86                          3.45     1480        1
4           4.320000  1.04                          2.93      735        1
..               …   …                           …      …      …
168         8.660000  0.74                          1.80      750        0
169         8.500000  0.67                          1.92      630        0
170         5.500000  0.66                          1.83      510        0
171         9.899999  0.57                          1.63      470        0
172         9.700000  0.62                          1.71      660        0

[173 rows x 15 columns]
```

[117]:
```
wines_df['Class_id'] = wines_df['Class_id'].replace(3, 0)
```

[118]:
```
msk = wines_df['Class_id'] == wines_df['Cluster']
msk.value_counts()
```

[118]:
```
True     172
False      6
dtype: int64
```

[120]:
```
Accuracy = (172/178)*100
print('Accuracy =', round(Accuracy,2), '%')
```

```
Accuracy = 96.63 %
```

3.5 K-means Clustering Model with PCA

We are going to try to evaluate the same model but by carrying out a Principal Component Analysis (PCA).

This is a dimensionality reduction method that simplifies the complexity of spaces with multiple dimensions while preserving their information. In other words, it allows "condensing" the information provided by multiple variables into just a few components.

After the analysis, we will be able to visualize how each of the clusters is distributed around each centroid. Additionally, we will compare the accuracy obtained with that of the previous model.

[91]:
```
pca = PCA(2, random_state=42)
wine_pca = pca.fit_transform(wines_subset_scaled)
```

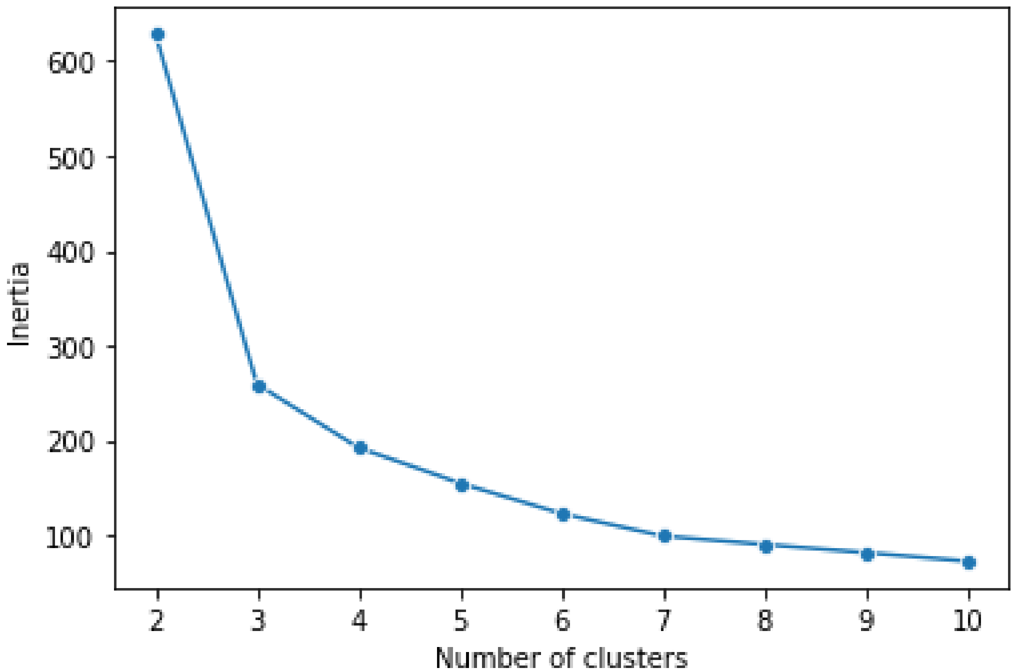
[121]:
```
wine_comps = pd.DataFrame(columns = ['comp_1','comp_2'], data= wine_pca)

wine_comps.head()
```

[121]:

	comp_1	comp_2
0	3.316751	-1.443463
1	2.209465	0.333393
2	2.516740	-1.031151
3	3.757066	-2.756372
4	1.008908	-0.869831

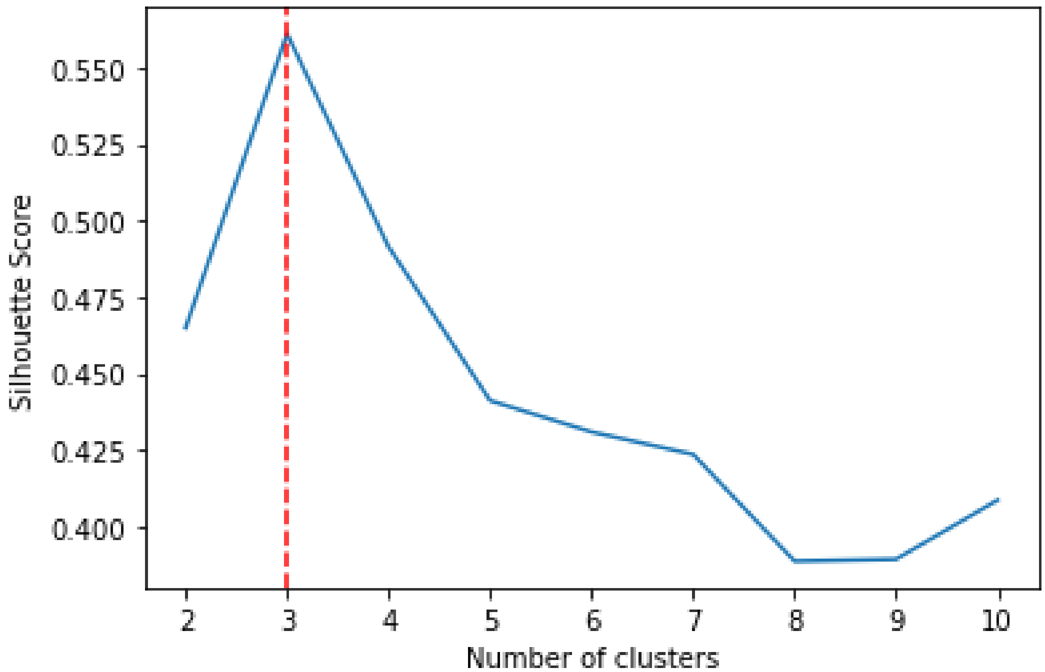
[138]:
```
Inertia_pca = kmeans_inertia(num_clusters, wine_pca)
```

[139]:
```
plot_pca = sns.lineplot(x=num_clusters, y=Inertia_pca, marker = 'o')
plot_pca.set_xlabel("Number of clusters");
plot_pca.set_ylabel("Inertia");
```



[140]:
```
sil_scores_pca = kmeans_sil(num_clusters,wine_pca)
```

[141]:
```
plot_pca = sns.lineplot(x=num_clusters, y=sil_scores_pca)
plot_pca.axvline(x=3, color='red', linestyle='--')
plot_pca.set_xlabel("Number of clusters");
plot_pca.set_ylabel("Silhouette Score");
```

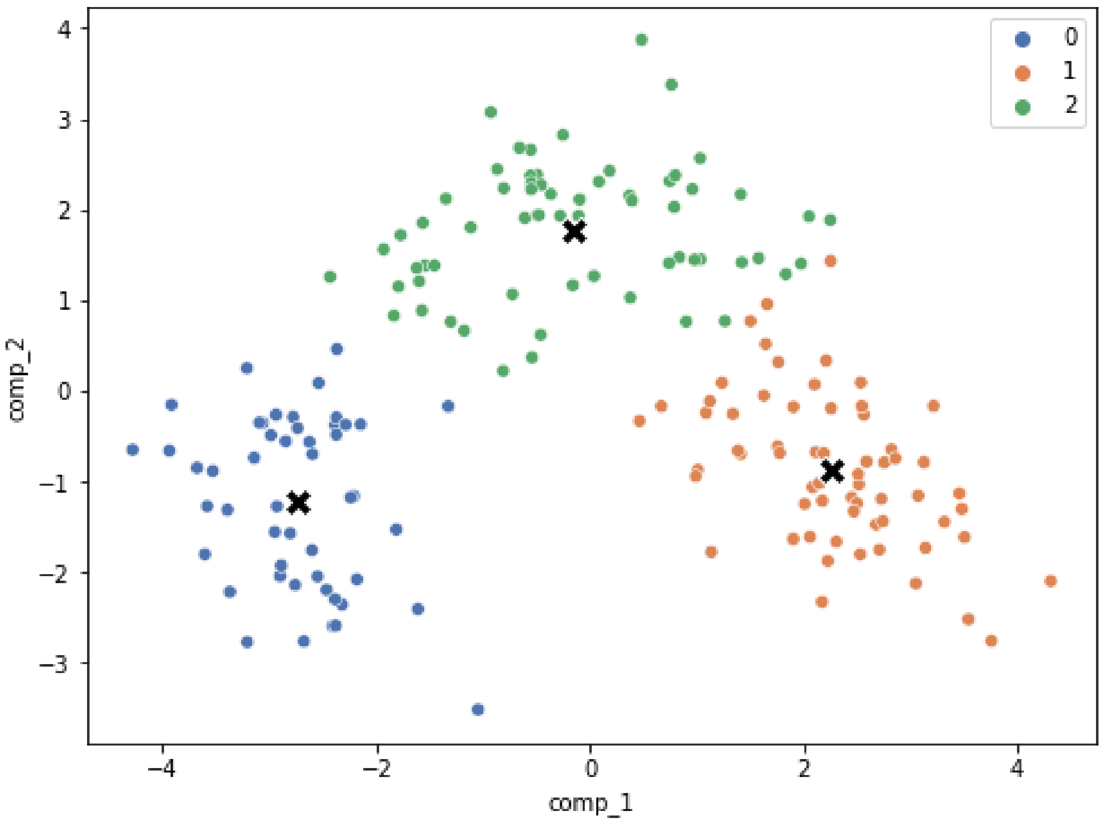


We can observe, like the previous case, we have our data partitioned into **three clusters**.

[122]:
```
KMeans_3pca = KMeans(n_clusters=3, random_state=42)
KMeans_3pca.fit(wine_comps)
```

[122]: KMeans(n_clusters=3, random_state=42)

[137]:
```
wine_comps['Cluster'] = KMeans_3pca.labels_

centroids = KMeans_3pca.cluster_centers_

centroids_comp_1 = centroids[:,0]
centroids_comp_2 = centroids[:,1]

plt.figure(figsize=(8,6))
sns.scatterplot(data=wine_comps, x='comp_1', y='comp_2', hue='Cluster',␣
 ↪palette="deep");
sns.scatterplot(x=centroids_comp_1, y=centroids_comp_2, marker='X',␣
 ↪c=['black'], s=150);
```



Lets verify if it assigned each point to the right group of wine, comparing with the first column removed at the beginning.

[132]:
```
Class_id = wines_df['Class_id']


concat_wines_comps = pd.concat([Class_id, wine_comps], axis=1)

concat_wines_comps.head()
```

[132]:

	Class_id	comp_1	comp_2	Cluster
0	1	3.316751	-1.443463	1
1	1	2.209465	0.333393	1
2	1	2.516740	-1.031151	1
3	1	3.757066	-2.756372	1
4	1	1.008908	-0.869831	1

[129]:
```
msk_pca = concat_wines_comps['Class_id'] == concat_wines_comps['Cluster']
msk_pca.value_counts()
```

[129]:
```
True     172
False      6
dtype: int64
```

[133]:
```
Accuracy_pca = (172/178)*100
print('Accuracy_pca =', round(Accuracy,2), '%')
```

```
Accuracy_pca = 96.63 %
```

4 Observation:

Both analyses show that our dataset is divided into three clusters. Since we had the solution to the problem, we were able to compare the results obtained by the models with the actual data, observing a grouping accuracy of 96.63% for both cases.